(1)

National Defence
Research and Development Branch

Défense nationale
Bureau de recherche et développement

TECHNICAL COMMUNICATION 90/304
April 1990

POST SCRIPT CODE FOR GREY-SCALED DISPLAYS

G.J. Heard

Defence Research Establishment Atlantic

Centre de Recherches pour la Défense Atlantique

Canada

UNLIMITED DISTRIBUTION

National Defence
Research and Development Branch

Défense nationale
Bureau de recherche et développement

POST SCRIPT CODE FOR GREY-SCALED DISPLAYS

by

G.J. Heard

April 1990

Approved by H.M. Merklinger
Head / Surveillance Acoustics Section

Distribution Approved by R.S. Walker

Acting Director / Underwater Acoustics Division

TECHNICAL COMMUNICATION 90/304

Defence Research Establishment Atlantic

Centre de Recherches pour la Défense Atlantique

Canada

ABSTRACT

PostScript is a versatile page description language used for the production of complex images combining graphics and text information. This note describes a PostScript program capable of producing a detailed grey-scaled image of information represented by a series of hexadecimal codes. A parameter section is included in the PostScript code that allows the user to modify the appearance of the diagram without the need to re-execute the program that generated the data and/or PostScript file in the first place. This code can be used on many different systems that support a PostScript printing device, but was originally intended for a 300 dot per inch laser printer. The PostScript code contains procedures to create time strings in hours, minutes, and seconds format from an input value in decimal hours.

RESUME

PostScript est un langage de description de page très versatile utilisé pour produire des images complexes combinant graphiques et textes. Ce rapport décrit un programme écrit en PostScript qui produit une image détaillée avec une échelle de gris, et représentée par une série de codes hexadécimals. Le code en PostScript inclut une section de paramètres qui permettent à l'utilisateur de modifier l'apparence d'un diagramme sans avoir à réexécuter le programme qui a généré les données et/ou le fichier PostScript original. Ce code peut être utilisé sur plusieurs différents systèmes qui supportent une imprimante de type PostScript, bien qu'il ait été conçu originellement pour une imprimante au laser avec une résolution de 300 points par pouce. Le code en PostScript contient des procédures qui transforment une donnée d'entrée de temps en heures décimales en une donnée de format ligne incluant heures, minutes, secondes.

Table of Contents

1. INTRODUCTION

This report describes a PostScript™ program that produces an annotated grey-scale plot of intensity versus time and an x coordinate. Intensity is represented by a maximum of 16 grey levels represented by the ASCII codes 0 to 9 and A to F. The ASCII codes are interpreted as hexadecimal with 0 representing white and F representing black. This report provides a very brief introduction to PostScript, discusses the program structure and provides a guide for changing the program code to tailor the program output for different applications.

PostScript, a page description language developed by Adobe Systems Incorporated, is an interpretive language that allows users to build up complicated graphic images, text, and line drawings into complete documents. PostScript is extremely flexible, and is capable of producing publication quality output for business, scientific and artistic applications. Many different implementations of PostScript exist for different types and models of printing engines. Not all "PostScript" printers are driven by legitimate system software; several clone and "look-alike" versions exist. The code described in this note executes properly on several different licensed PostScript printers including Apple and QMS printers.

Programming in PostScript is a strange experience at first for most users. PostScript is a stack based language which employs 'post-fix' notation. Groups of characters in the input stream are called objects. Objects have various attributes; some are value objects, some are executable, and others are string or array objects. Using the language is similar to using a 'reverse Polish' calculator. For example, to add two numbers the following code segment could be used:

A B add ,

where A and B are the addends (value objects) and add is the operand (executable object). The addends are removed from the stack, summed and the result of the addition is pushed onto the stack. The stack is a first-in-last-out (FILO) structure. Only the last 'pushed' or entered value is immediately accessible. This value is sometimes said to be 'on top of the stack'. Each enter operation may be visualized as causing the current stack contents to drop one level deeper into the stack.

The only net difference between the PostScript interpreter and the 'reverse Polish' calculator is that PostScript performs an automatic 'enter' every time it encounters an object in the input stream. In this case, the objects that are automatically 'enter-ed' are the addends. PostScript also differs in that after 'enter-ing' the addends it continues to scan the input stream and then encounters the object **add**. This object is executable. The interpreter fetches the **add** procedure from a storage area called a dictionary and executes the procedure. For both the calculator and PostScript the result of the addition is left on the stack and becomes the value that is directly accessible to further commands or functions. The calculator displays this value directly on its display, but the PostScript result is not displayed anywhere; it is only a bit pattern in the printer's memory. The programmer may or may not wish to actually print the result. Usually, such an operation produces an intermediate value used in setting up a font or graphic image. Occasionally the result may be the actual desired value to be displayed on a piece of paper; in this case, the programmer must take action to cause this result to be printed. To print the result the programmer must specify the font type and size, the position and orientation, and the colour or grey level with which it is to be printed. The following sequence would cause the result to be displayed:

```
/line 80 string def                          % create a string object
/Helvetica findfont 12 scalefont setfont     % define the font and size
72 72 moveto                                 % set the currentpoint
10 20 add                                    % the addition operation
line cvs                                     % convert result to a string
show                                         % stroke the currentpath
showpage                                     % transfer image to paper and eject sheet.
```

In the above example the first line creates a string object (in this case a variable) called **line**. Eighty bytes of storage are assigned to **line**. The percent (**%**) sign indicates a comment field; everything following the **%** up to the end of the line is ignored. The second line of the example specifies that a 12 point Helvetica font is desired. The **moveto** command working in units of points (defined as 1/72 inch, almost the same as a printer's point 1/72.27 inch) sets the location (x, y position) of the resulting image (or path in PostScript jargon) relative to a pre-determined origin. As the **moveto** command is executed, it removes its parameters (i.e. x=72, y=72) from the stack. The origin of the diagram is initially set to the bottom left hand corner of the page. This origin may be translated and the coordinate system rotated by PostScript commands. The current origin and orientation are stored in a set of variables describing the current graphics state. In the above example, the orientation is defaulted to whatever is in the current graphics state. A portrait layout, defined as the page being oriented with the narrow edge horizontal, is the usual default. The interpreter then encounters the addends, which in this case are 10 and 20. The addends are placed, one above the other, on the stack. The executable object **add** is then encountered. The **add** command removes the top two stack entries (10, 20), sums them and places the result on top of the stack. When the interpreter parses the next line of code it encounters the string object called **line** which is placed on top of the stack pushing the previous stack contents one level down. The interpreter then encounters the executable object **cvs** which causes the result of the addition (i.e. 30) to be converted to ASCII codes which are stored in a subset of the space reserved by the string object **line**. The net effect is that the result of the addition and **line** are removed from the stack along with the executable object **cvs** and replaced by a string object containing an ASCII representation of the addition result. The effect is similar to an internal formatted write statement in FORTRAN; the binary result is converted to ASCII that a human can easily understand. Note though, that the result is written into a string variable, not a file, or printed out on a page. The next line of code contains the **show** command. **Show** causes the 'currentpath to be stroked'; this is PostScript jargon that means that the image (or path) is transferred (stroked) into a memory map of the page about to be printed. So far nothing has been marked on the paper. The last object **showpage** transfers the memory image to the paper and causes the sheet to be ejected from the printer.

The above example is intended only to initiate readers to PostScript. It will probably have frightened most potential users away, but keep in mind that the complexity of PostScript is what allows it to be used to construct general images. With practice, PostScript's 'post-fix' notation and stack operation become very convenient. A full description of the actions above would fill many pages and involve discussion of structures well beyond the scope or intent of this report. In order to be able to modify the code described in this note, references 1 to 4 will be found to be very helpful.

The most common examples of PostScript Engines at DREA are the Apple LaserWriter printers. These printers are in common usage by the many Macintosh computer users at DREA. It may seem strange to some readers to discover that the Apple NTX printer they use daily is actually a computer more powerful than the MacPlus or MacSE on their desks. In fact, the NTX printer contains a 68020 processor and at least 2 megabytes of RAM. This powerful printer is required because PostScript is a memory-hungry and computationally intensive program. A large amount of memory is required for the bit-map of the page, the input stream, the fonts, the dictionary, and

any incomplete path (or image) definitions. Fast number crunching is required when one considers the thousands of rotations, scalings, translations and other operations required to print an image that may contain several million pixels. This is quite a foreign concept to the programmer who is used to dumb printers driven directly by a host microcomputer (some PostScript implementations are like this too). In general the PostScript printer is a power computer, fully capable of doing all the calculation, processing, and display of results.

Most PostScript printers are not used as the primary computational device; usually the printer is sent a cryptic, compressed ASCII stream which contains as much of the final result as is possible. The printer is generally only responsible for the graphic and font manipulation required to put the image on the page. This philosophy has several benefits, prominent among which are faster printing times. The code described in this note does not make full use of this philosophy, partly because it was written by a novice PostScript programmer and partly because of a desire to off-load the computer generating the output. Also, with the PostScript code performing many of the calculations, changes can be made to the parameter section of the code and the desired result obtained without the need to use the large computer that originally generated the picture. The next section will describe the code and explain in part the operation of the various components.

Several options exist in the way printing may be carried out, depending on what hardware/software is available to the user. The most accessible methods at DREA will be discussed in section 3. Small modifications to the code which tailor it for slightly different applications will be discussed in section 4. Finally, the code's application and capabilities will be summarized in section 5.

2. SOURCE CODE DESCRIPTION

In this section, the grey-scale program source code will be discussed. A detailed description of each procedure will not be given, rather just a general overview of those procedures which most often require modification for other purposes will be considered. The complete source code listing will be found in Appendix A. This section will repeat only those code segments under discussion.

To begin the discussion of the source code we shall first consider the program structure. PostScript programs are highly structured. In fact, a specific format has been agreed upon, and programs written with this format are said to be conforming. Many PostScript programs do not conform to the standard. Such programs may encounter difficulties when transported to different computer systems. The code presented in this note conforms at least partially to the standard and has been found to run without modification on three different computer systems employing four different PostScript printers. Listed below are the statements that define the structure of the PostScript program.

```
%! PS-ADOBE-1.0
%%TITLE: GREY-SCALE HALF-TONE DISPLAY
%%CREATOR: G.J. HEARD FROM SASPEX
%%CREATIONDATE: Mon Jan 29 15:06:19 1990
%%FOR: Jimbob
%%PAGES: 2
%%BOUNDINGBOX: 10 10 600 780
%%ENDCOMMENTS
save

.
.
PROLOGUE
.
```

```
%%ENDPROLOG
%%PAGEFONTS: Helvetica
%%PAGE: 1 1
.
.
                    BODY
.
.
%%TRAILER
showpage
restore
.
.
              COVER PAGE PROGRAM
.
.
```

The percent (%) sign indicates a comment field in PostScript. Any characters on the line following the % sign are ignored by the PostScript scanner. Some editors and Macintosh applications cause problems with long comments by breaking lines, placing part of the comment on a separate line. This effect can be quite a nuisance when moving the code between systems and must be monitored closely.

The first two characters of the PostScript file (%!) are called the magic number. The magic number is a key that allows printer scheduling software to pass the PostScript program as a program rather than just listing the code on paper. Note that the magic number is followed by a space. This space is absolutely necessary. Programs executed under some pre-1990 software releases will not work without the space, despite the fact that all examples in Adobe publications omit it.

Double percent signs (%%) indicate a conforming comment. Conforming comments define the code's structure. PostScript does not interpret the comments at all; however, the printer driver software or the printer itself may interpret some or all of the conforming comments. An introductory explanation of these comments can be found in reference 1.

The first group of comments up to and including **%%ENDCOMMENTS** are referred to as the header. Following the header is a user-written code segment that consists of procedures which remain unchanged from execution to execution. This user procedure section is called the prologue; it includes the **%%ENDPROLOG** comment. From the "**%%PAGEFONTS: Helvetica**" comment up to **%%TRAILER** is the body portion of the program. The body consists mainly of data and parameters and the main procedure which calls the prologue procedures to accomplish the desired task. Following the body is the trailer section. The trailer generally includes any clean-up tasks required to restore the printer's original state before your program executed. In our example, the grey-scale program is followed by a non-conforming program that generates a cover page for the grey-scale picture.

Some of the most important code for producing a grey-scale diagram will now be considered. The choice of pixel sizes and grey-levels depends strongly on the application. The interaction of these parameters with the PostScript imaging scheme is complex, and a somewhat empirical approach has been found to produce the best results. Each of references 1 to 4 discusses the use of the image command and each leaves questions unanswered, fails to point out the PostScript problems, and occasional misleads the reader. The code segments presented here have served well for over a year, but they also do not answer all the questions that arise when

developing a PostScript grey-scaling application.

The following code segment allows the user to select the shade of grey assigned to each level.

```
/transarray
[
  100 97 97 90 90 80 80 75 70 65 60 50 43 23 12 0 0 % grey levels
] def
{
  16 mul cvi
  transarray exch get
  100 div
}
settransfer
```

This code was modified from reference 4. Since the current application calls for a 4x4 device pixel grey-scale element, seventeen levels are possible. Due to the 'splash' or spill over of the individual device pixels it is only possible to create approximately 10 distinct grey levels with a 4x4 pixel and an Apple LaserWriter. The above code assigns a percentage of white to each level. The percentage has been determined empirically and has been found to produce a grey-scale which the human eye perceives as almost linearly increasing in density until the higher levels are reached, after which the density appears to increase more rapidly. Figure 1 shows the grey-scale obtained with this transfer function.

In this application, the hexadecimal codes 0-F are used to represent the different grey levels; hence, only 16 grey levels are used of the 17 available. This explains why the last two levels are set completely black in the above defined array object **transarray**. **Transarray** also assigns the same percentage of whiteness to three pairs of the possible 17 levels, leaving a total of 13 grey levels. This has been done because it was desired to make the lower grey levels appear to darken in a linear fashion and because it is not possible to accurately reproduce all 17 levels distinctly.

The last point to be discussed with reference to Fig. 1 is the relatively large jump in whiteness between the first two levels. The first level is 100% white and means the printer does not mark the page. The second level is specified as being 97% white, yet in fact, the printer output is probably closer to 75% white. This undesirable jump appears to be a limitation of the PostScript grey-scaling process. Smaller jumps are possible, but only at the expense of a larger grey-scale element which cannot be tolerated in the present application. One possibility for future improvement of the grey-scale program lies in the 'screen' definition. Experiments with the existing grey-scale program show that to some extent an individual pixel's appearance on a piece of paper depends upon its environment. That is, the size of the pixel appears to be altered by the level of the pixels near it. Whether this effect arises from the electrical properties of the print engine, or whether it arises because of the software, either the grey-scale program or the PostScript system itself, is at present unknown. The magnitude of this effect can be seen by comparing the grey-scale element sizes in Fig. 1 against the corresponding levels in Fig. 2. The lightest elements in Fig. 2 are represented by a much smaller dot than that for the same level in Fig. 1. The only difference in the production of Figs. 1 & 2 is that different image data was used.

The next code segment defines the PostScript screen and spot function:

```
75 45
{
  abs exch abs 2 copy add 1 gt
```

```
    { 1 sub dup mul exch 1 sub dup mul add 1 sub}
    {dup mul exch dup mul add 1 exch sub }
  ifelse
}
setscreen .
```

The **setscreen** function defines the spot size, the angle of the spots, and the manner in which device pixels are turned on to create a grey-scale element. In the above code a spot frequency of 75 and an angle of 45° have been chosen. The spot function was taken from reference 4, and is said to be the default. Some of our older printers do not appear to use this function so it has been explicitly requested to ensure uniformity of product. The spot function illustrated is based on an **ifelse** choice which causes a black spiral dot to grow from the centre for low density elements and a central white dot to shrink for high density elements. This trick is used to obtain more-saturated blacks.

A more complete explanation of the screen can be found in references 1-4. The screen can be visualized as a wire mesh through which the image is sprayed onto the paper (an analogy with the early mechanical means of semi-toning). The PostScript manual says that the screen function rarely needs to be changed; the manufacturer installs a default screen function that works well with the printer. It is true that the screen function supplied as default works quite well; however, it does possess a fairly serious drawback in that the resultant screen images exhibit a marked striation pattern. This pattern probably arises from the use of a screen frequency parameter corresponding to fewer grey-scale elements per inch than is possible on the basis of the print device resolution.

PostScript always functions in a fixed coordinate space based on points (1/72"). Depending on the actual printing device, device pixels may be larger or smaller than 1 point. The PostScript interpreter fits the image to the available print device resolution as best as it can. Users generally want to think in terms of some other coordinate system that is more native to their particular problem. The following code segment defines procedures based on a simple linear transformation that converts user coordinates to points.

```
/xms {xu xl sub} def
/yms {yu yl sub} def
/xuc {xl sub xms div} def
/yuc {yl sub yms div} def
```

The first two lines above define the differences between the x and y limits, the next two lines define procedures **xuc** and **yuc** which convert the current x and y values to ratios of the total distance across the x and y axes respectively. The reason for this is that PostScript treats all images as being 1 point by 1 point regardless of the actual size of the printed image. The required transformations from user units to PostScript units are easily seen to be linear transformations which map the individual axis ranges into the fixed range 0 to 1.

Related to the transformation of the user coordinates to the system coordinates is the axis scaling. The scale is changed in numerous places in the grey-scale program. Scale changes are used to allow the tick marks to be drawn at the appropriate locations. The scale is also used to control the size of the final diagram. An implicit scaling is carried out in the current version of the grey-scale program; **cpi**, or characters per inch, controls the net resolution of the printer output. At present **cpi** is defined as

```
/cpi {100 div} def ,
```

where the 100 implies the use of 100 grey-scale elements per linear inch of paper. The inclusion of this statement causes a 33% reduction in the final image size. The net result is a slight degradation

in the final image quality, but it allows 1024 grey-scale elements to be displayed on an 8.5x11 inch piece of paper oriented in the landscape (wide orientation) format. The effect of the scaling can be removed by changing the 100 in **cpi**'s definition to 75. This change will size the image to match the 300 dot per inch laser printer output with a 4x4 grey-scale element.

The next code segment to be discussed creates the labelled ticks for the x and y axes of the diagram. It is by the far the weakest part of the program, since it does not respond properly to changes in scale and is not perfect with respect to the variability of the finished size of the diagram. Future improvements could certainly remove redundancy, device dependency and other problems; however, the code has survived the production of many thousands of diagrams, and given current priorities, will probably endure some time yet.

```
/xaxis {
  newpath
  xstart xinc xu
  {
    dup dup
    xuc 0 moveto
    xuc -4 ylen div lineto
    dup xuc 1 moveto
    xuc 4 ylen div 1 add lineto
  } for stroke
} def

/yaxis {
  newpath
  ystart yinc yl
  {
    dup dup
    yuc 0 exch moveto
    yuc -4 xlen div exch lineto
    dup yuc 1 exch moveto
    yuc 4 xlen div 1 add exch lineto
  } for stroke
} def

/box {
  newpath
  -4 -4 moveto
  0 ylen 8 add rlineto
  xlen 8 add 0 rlineto
  0 -1 ylen mul 8 sub rlineto
  closepath
} def

/xlabs {
  /Helvetica findfont 12 xnum div scalefont setfont
  xlen ylen scale
  xaxis
  1 xlen ylen div scale
  xstart xlskp xinc mul xu
  {
    dup label cvs
    dup length -2 xlen div mul
```

```
    3 -1 roll
    xuc
    add -18 xlen div moveto
    show
  } for
} def

/ylabs {
  /Helvetica findfont 12 ynum div scalefont setfont
  1 ylen xlen div scale
  yaxis
  ylen xlen div 1 scale
  ystart ylskp yinc mul yl
  {
    dup repack
    -3 ylen div exch yuc add -40 100 cpi mul ylen div
    exch moveto
    time
    show
  } for
} def
```

Five procedures are defined by the above code segment (**xaxis**, **yaxis**, **box**, **xlabs**, **ylabs**). The **xaxis** and **yaxis** procedures are responsible for drawing the tick marks on the axes drawn by the **box** procedure. The last two procedures, **xlabs** and **ylabs**, call **xaxis** and **yaxis**, and convert the time and the x coordinate information to appropriately formatted and positioned strings. A great part of the effort in these routines is maintaining an appropriate scaling during each of the operations. This effort could largely be done away with through more use of the **gsave** and **grestore** features of PostScript. The procedure **ylabs** makes use of a procedure called **repack**; this procedure is not reproduced here but can be found in the source code listing in the appendix. The purpose of the **repack** routine is to convert the decimal hours time information to an hours, minutes, seconds format with colons dividing the sections.

The following section of PostScript code is perhaps the most interesting to the user. This section of code defines all the parameters that the PostScript program requires for execution. This section is often modified with an editor when it is desired to change the size of the diagram, alter the axes or labelling, or rotate the diagram.

```
/x     1.0000 def
/y     0.6000 def
/theta     0.0000 def
/xnum   512 def
/ynum   100 def
/scf     1.0000 def
/xl     0.0000 def
/xu   127.7484 def
/yl   11.4942 def
/yu    11.4392 def
/xstart    20.0000 def
/xinc    20.0000 def
/ystart    11.4392 def
/yinc     0.0055 def
/xlskp   2 def
/ylskp   2 def
```

In the above code segment, **x** and **y** are the coordinates of the diagram's origin (defined as the lower left hand corner of the grey-scale image) relative to the lower left hand corner of the page. The coordinates are given in inches and are independent of the choice of orientation specified by **theta**. **Theta**, measured counter-clockwise, is given in degrees with 0° corresponding to a portrait orientation (x axis of figure aligned with the narrow edge of the paper). The next two parameters tell PostScript how many grey-scale elements there are in the x and y directions. The amount of image data included in the file must be accurately described by the product of **xnum** and **ynum**. One restriction of the PostScript image command is that **xnum** and **ynum** must both be positive, even integers. The parameter **scf** is the scale factor. A scale factor of 2 causes the length of each axis to be doubled with a resulting 4 times increase in the diagram area. Text information does not respond appropriately to the scale factor. The next four parameters (**xl**, **xu**, **yl**, and **yu**) describe the lower and upper limits of the x and y axes. Note that the diagram is usually arranged so that the y axis represents time and that the lower y boundary is a later time than the upper y boundary. All times are specified in decimal hours. The next two parameters **xstart** and **xinc** describe where the first x axis tick should be placed and the increment to the next tick mark. Similarly, the next two parameters **ystart** and **yinc** fulfil the same function for the y axis. Finally, the last two parameters **xlskp** and **ylskp** tell PostScript how many tick marks to skip over (including the current tick) before labelling another tick. In the example above, every other tick is labelled.

Figure 2 is a sample output of the grey-scale program. The figure axes have been given arbitrary ranges and the image data itself has been arbitrarily built up of the ASCII codes 0-F. In this example, the parameter section listed below was used.

```
/x    2.0000 def
/y    5.000 def
/theta    0.0000 def
/xnum   512 def
/ynum   192 def
/scf    1.0000 def
/xl    0.000 def
/xu   127.7484 def
/yl   11.5 def
/yu    11.00 def
/xstart    20.0000 def
/xinc    20.0000 def
/ystart    11.0 def
/yinc    0.083334 def
/xlskp   2 def
/ylskp   2 def
```

In the example above, the parameters **xnum** and **ynum** describe the amount of image data in the completed diagram. Both **xnum** and **ynum** must be positive, even, integers. The parameter **xnum** describes the number of grey-scale pixels in the x coordinate direction, while **ynum** describes the number of pixels in the y coordinate direction.

The following code segment, is responsible for position, sizing and producing the grey-scaled image:

```
theta rotate
x inch y inch translate
gsave
xlen ylen scale
```

```
xnum ynum
4
[xnum 0 0 -1 ynum mul 0 ynum]
{currentfile picstr readhexstring pop}
image
.
.
.
grestore .
```

The first line of the above listing causes the coordinate system to be rotated counter-clockwise by **theta** degrees. The second line of code positions the lower left hand corner of the image at the desired (**x**,**y**) coordinate. The command **gsave** is used to protect the existing graphic state from the scaling operations in the code up to the **grestore** command. Following **gsave** is an explicit **scale** change which causes the image produced to measure **xlen** by **ylen** points. The axes lengths **xlen** and **ylen** were computed previously from the parameters **xnum**, **ynum**, **scf**, and the implicit scaling operation **cpi**. The next four lines of the above code segment are arguments for the PostScript **image** command. The first two arguments are simply the number of pixels in the x and y directions (i.e. **xnum** and **ynum**). The third argument, 4, describes the number of bits per grey-scale pixel. The fourth argument is a matrix object that specifies a coordinate transformation from pixel space (**xnum** pixels x **ynum** pixels) to the (1 point x 1 point) image space. The fifth and final argument is a procedure that is executed repeatedly until **xnum** times **ynum** pixels have been presented to the **image** command. The procedure must obtain the image data from somewhere and leave a string containing an arbitrary length sub-set of the image data on top of the stack. In this case, the image data is placed immediately after the **image** command and the procedure reads the input stream and filters out all the ASCII codes except for 0-9, a-f or A-F. The remaining filtered characters are interpreted as hexadecimal codes and represent the grey-scale levels. White is represented by a 0 and black by an f or F. The actual image data is organized such that the first hexadecimal code following the **image** command represents the grey level of the image pixel in the output diagram's upper left hand corner. Each succeeding character describes the grey level of the character to the right of the previous character until **xnum** characters have been processed. The character **xnum**+1 characters into the file, represents the left-most character of the second pixel row from the top of the diagram. This character interpretation continues until **xnum** times **ynum** characters have been processed. The character **xnum** times **ynum** characters into the file represents the grey level of the lower right hand pixel. The image codes may be written to the file in any organization that meets the constraints mentioned above. It is convenient to separate the pixel rows by carriage returns; for example, if **xnum**=512 and **ynum**=100, then a 100 rows of 512 characters would be a sensible organization scheme. You could equally well place all 51200 characters on a single line, or have 51200 lines of 1 character each. This choice is left to the user. Although the **readhexstring** function will strip non-valid hexadecimal codes from the input stream, it is wise to avoid any non-valid characters except for carriage returns. This warning also applies to **%**, which the user might be tempted to use to provide a comment area describing the image. The above procedure doesn't recognize the **%** and such a comment would be interpreted as image data. Any mismatch between the product of **xnum** and **ynum** and the total number of hexadecimal codes available will result in a failure to produce a diagram.

3. USING THE CODE

The Macintosh user may present the code in several different ways to a PostScript printer; this section describes the simplest way of producing a diagram. A second method, which may be available to some is to copy the code into Microsoft's Word and give it a PostScript type. This procedure is described in the Word user reference manual and in several other publications. This procedure takes more effort and may cause problems because the current grey-scale routine does not define its own dictionary; therefore, it relies on there being enough dictionary space in the printer after the Word PostScript glossary has been loaded.

A simpler method of printing the diagram is to obtain a copy of Adobe's SendPS program. This Macintosh application was kindly made public domain by Adobe and is a small program that can talk directly with a PostScript printer. To print the file, simply start up the SendPS program and choose 'Down load file' from the file menu. Select the desired file in the dialogue box and SendPS will create a window showing the transfer status. If errors are encountered SendPS saves the printer's diagnostic information in a file which can be later reviewed with an editor.

Current versions of the SendPS program do not support background processing. This may cause the user some annoyance if many diagrams are to be printed. The program PrintPS is currently under development by S. Hughes of DREA that will allow background processing and the submission of multiple files for printing.

On the Concurrent Unix machines, the print spooler software, Transcript, will pass the file to the printer appropriately if the magic number **%!** is included as the first two characters in the file. The user may use any of the UNIX system print request submission commands such as enscript or lp.

For users of the Surveillance Acoustic section VAX cluster, the global 'psplot' is defined to be a command routine that will submit a file to the DEC Scriptwriter. Simply type 'psplot filename' followed by a carriage return. The file referred to by 'filename' will be submitted to the print queue.

Users of other systems will also be able to use this grey-scaling routine, but the details of the submission will depend on the system specifics. Most systems with a print spooler and a PostScript printer will recognize the magic number **%!** and handle the file properly.

4. MODIFYING THE CODE

This section is intended to provide first time users with guidelines for modifying the code in minor ways. It is not intended to describe how major changes could be implemented.

The simplest changes that can be made are carried out in the parameter section described earlier. The most often desired changes are made to the axis tick start position and spacing, and the associated labelling. This type of change is carried out very simply with an editor. Simply change the **xstart**, **xinc**, **ystart**, **yinc**, **xlskp**, and **ylskp** parameters as desired. Recall that the y axis represents time (usually) and that all times are entered to arbitrary precision in decimal hours. As long as the **xl**, **xu**, **yl**, and **yu** parameters are left unchanged, the routine will properly locate the ticks and labels at the times requested. Notice that no error checking is performed by the code, so impossible requests will usually result in garbage output.

Another frequently desired change involves the size of the diagram. Size changes are accomplished by changing the **scf** parameter. The text labels that result after a scaling operation are usually misplaced. This error can be corrected by either eliminating the labelling completely or by adjusting the **xlabs** and **ylabs** routines. Since the adjustments to **xlabs** and **ylabs** can be

complicated, and really indicate that these routines need to be re-written, it is usual to eliminate the labelling. To do this one has to remove the show command from the **xlabs** and **ylabs** procedures. This technique is illustrated for **ylabs** below.

```
/ylabs {
   /Helvetica findfont 12 ynum div scalefont setfont
   1 ylen xlen div scale
   yaxis
   ylen xlen div 1 scale
   ystart ylskp yinc mul yl
   {
      dup repack
      -3 ylen div exch yuc add -40 100 cpi mul ylen div
      exch moveto
      time
      %show
      pop
   } for
} def
```

First, the **show** command was removed by using the comment character **%**. In addition to removing the **show** command it is necessary to remove **show**'s argument from the stack; the **pop** command does this. This example illustrates an important point with PostScript code modifications. Not only must the input stream command be removed, but the stack must be treated to remove the arguments that would have been processed by the removed commands.

Some applications of this code may not require the same grey-scale levels provided by the **transarray** definition. Changing the first 16 numbers in the **transarray** definition will allow the user to modify the resulting grey levels. Recall that **transarray** contains the percentage of white in the level. A completely white pixel is described by an entry of 100 and a completely black pixel by an entry of 0. More levels can be incorporated by changing the 16 in the line '**16 mul cvi**' to the desired number and adding the extra elements to the **transarray** definition. This will not be effective unless the grey-scale element size is altered by changing the appropriate image command parameter.

Some applications of the code may require a change to the spot function in the **setscreen** call. An active sonar display could be emulated with the current code, in which case a more appropriate spot function would be such that the grey element darkens by adding pixels to horizontal lines growing from the bottom of the pixel upwards. For example the procedure

```
{exch pop}
```

would prioritize device pixels on the basis of their vertical position within the grey scale element. Such spot functions must return a value between -1 and 1, derived from the pixel coordinates within the grey-scale element. The origin of the coordinate system used here is located at the centre of the grey element and extends from -1 to 1 in both the x and y directions. The spot function is called with the device pixel coordinates on the stack. The example spot function above throws the x coordinate away and utilizes only the y coordinate. Pixels with the same priority are arbitrarily picked by the interpreter to correspond to the different grey levels.

Additional functions could be added to include various titles or other devices in the final diagram. Such add-on changes are best surrounded by **gsave** and **grestore** statements so as to leave the current environment undisturbed. Make certain that the stack before and after execution of the add-on functions is the same. An example of adding a title is given below.

```
.
        previous unaltered code
.
gsave
  /Times-Roman findfont 18 scalefont setfont
  3 inch 1 inch moveto
  (This is the title text.) show
grestore
.
        previous unaltered code
.
```

In this example, the **gsave** and **grestore** commands protect the so called graphic environment. The code following selects a new font, moves to the desired location, and '**show**s' the text. The previously defined scaling, orientation and other parameters are used in the example.

5. SUMMARY

The PostScript code presented in this note has made possible the production of many grey-scaled diagrams. Originally intended for a single purpose, it has been pressed into service for several different applications requiring a grey-scaling capability. This note has described the structure of the program and provided some description of the operation of the various procedures. A complete source listing has been included in the appendix. Some of the bugs encountered in producing this code have been described and some suggestions for future improvements have been given. Brief instructions have been given to users of this program that should help them obtain their finished diagrams.

APPENDIX A - SOURCE CODE LISTING

The following listing presents the PostScript grey-scaling routines in use at DREA. Only the information contained in the body and cover page sections of the code is changed for the production of different diagrams. The parameter section and the hexadecimal codes are generated by another program which previously: 1) creates an output file containing the header section, 2) opens a data file containing the PostScript prologue and appends this information to the header section, 3) then appends the parameter and image data, and 4) finally appends the trailer section and the cover page routine. An example of a program that uses this technique is described in reference 5.

```
%! PS-ADOBE-1.0
%%TITLE: GREY-SCALE HALF-TONE DISPLAY
%%CREATOR: G.J. HEARD FROM SASPEX
%%CREATIONDATE: Mon Jan 29 15:06:19 1990
%%FOR: Jimbob
%%PAGES: 2
%%BOUNDINGBOX: 10 10 600 780
%%ENDCOMMENTS
save
/transarray [
  100 97 97 90 90 80 80 75 70 65 60 50 43 23 12 0 0 % this is the 75 screen
] def
{
  16 mul cvi
  transarray exch get
  100 div
}
settransfer
75 45
{abs exch abs 2 copy add 1 gt
     {1 sub dup mul exch 1 sub dup mul add 1 sub}
     {dup mul exch dup mul add 1 exch sub}
  ifelse
}
setscreen
/Helvetica findfont 12 scalefont setfont
/inch {72 mul} def
/cpi  {100 div} def
/xms {xu xl sub} def
/yms {yu yl sub} def
/xuc {xl sub xms div} def
/yuc {yl sub yms div} def
/xaxis {
  newpath
  xstart xinc xu
     {dup dup xuc 0 moveto
      xuc -4 ylen div lineto dup xuc 1 moveto xuc 4 ylen div 1 add lineto
     } for
  stroke
} def
/yaxis {
  newpath
  ystart yinc yl
```

```
    {
      dup dup yuc 0 exch moveto
      yuc -4 xlen div exch lineto dup yuc 1 exch moveto
      yuc 4 xlen div 1 add exch lineto
    } for
  stroke
} def
/box {
  newpath
  -4 -4 moveto
  0 ylen 8 add rlineto
  xlen 8 add 0 rlineto
  0 -1 ylen mul 8 sub rlineto
  closepath
} def
/label 12 string def
/time 9 string def
/grab { dup floor dup 3 1 roll } def
/convrt { sub 60 mul } def
/break {grab convrt grab convrt floor cvi 3 1 roll cvi exch cvi } def
/build {
  dup 10 lt
    {time 0 48 put label cvs time 1 3 -1 roll putinterval}
    {label cvs time 0 3 -1 roll putinterval}
  ifelse
  time 2 58 put dup
  10 lt
    {time 3 48 put label cvs time 4 3 -1 roll putinterval}
    {label cvs time 3 3 -1 roll putinterval}
  ifelse
  time 5 58 put
  dup 10 lt
    {time 6 48 put label cvs time 7 3 -1 roll putinterval}
    {label cvs time 6 3 -1 roll putinterval}
  ifelse
} def
/repack {break build} def
/xlabs {
  /Helvetica findfont 12 xnum div scalefont setfont
  xlen ylen scale
  xaxis 1 xlen ylen div scale
  xstart xlskp xinc mul xu
    {dup label cvs dup length -2 xlen div mul 3 -1 roll
    xuc add -18 xlen div moveto show}
  for
} def
/ylabs {
  /Helvetica findfont 12 ynum div scalefont setfont
  1 ylen xlen div scale
  yaxis
  ylen xlen div 1 scale
  ystart ylskp yinc mul yl
    {
    dup
```

```
      repack
      -3 ylen div exch yuc add -40 100 cpi mul ylen div exch moveto
      time show
    } for
} def
%%ENDPROLOG
%%PAGEFONTS: Helvetica
%%PAGE: 1 1
/x    1.0000 def
/y    0.6000 def
/theta    0.0000 def
/xnum   512 def
/ynum   100 def
/scf    1.0000 def
/xl    0.0000 def
/xu   127.7484 def
/yl   11.4942 def
/yu   11.4392 def
/xstart   20.0000 def
/xinc   20.0000 def
/ystart   11.4392 def
/yinc    0.0055 def
/xlskp  2 def
/ylskp  2 def
1 setgray
theta 45. ge
   {
   8.4 inch 0 inch translate
   90 rotate
   1 inch 8 inch moveto
   (User:JimBob/SEQ.#:   1/Mon Jan 29 15:06:19 1990/RES.=   0.25000 HZ)show
   -90 rotate
   -8.4 inch 0 inch translate
   }
   {
   1 inch 10.5 inch moveto
   (User:JimBob/SEQ.#:   1/Mon Jan 29 15:06:19 1990/RES.=   0.25000 HZ)show
   }
ifelse
theta 90. eq { 8.4 inch 0 inch translate } if
/xlen xnum inch cpi scf mul def
/ylen ynum inch cpi scf mul def
/picstr xnum string def
theta rotate
x inch y inch translate
gsave
xlen ylen scale
xnum ynum 4 [xnum 0 0 -1 ynum mul 0 ynum]
{currentfile picstr readhexstring pop}
image
F09597696C9689A6013052955D00308477CFFA8B395DA4A53699D70604D8B720F894BAAF
F9FD87FBF90085A85C0F98FF7B70AC17EFAFDC582AE0707168668C300CA928B83B60D8
8BF4DEDEF6C7FD182BFFF9F9308458890676F1990B1DA2FFFBD091228ECADFC0CAB6F
C3BFF5F6F4FC7ADFCA07069083C104ECFB5FC39F3FE9D5FDEDA956B88047CF08A9907
```

74A8AD69769D0A4BEEFFAFB07FFBAE3B85A69D7429B5637FF8FF9FFBDF6AFAB198538
4903CBA7FECFE9F9D9BA6899900717057D7D0FECCEF3DA98AB7190C6008889A4DADAC
5CF77BDC709C960003208645424ABFFFFBCBEC62755118C9749A859B80B4000A3D67B9
A03004400038O100669635B6

.
.
.

98 MORE LINES OF PICTURE DATA

.
.
.

FFFFFDFFFAFD6F7C5F0560064604999339F6BFC90B38C5048509575448840 36ECB0E1B9
EFFFFE7D56B5BDD6EC726246273DB73CDFFFFC6F0E4C7E1D7BC605288B0D08600344B
4006F754C96627C98787FF6CEE87139470A4DDF8DF8483D0EEFFFF9FCDCFA497B555885
C5F87AFFBAFFF9FCDE008E31C081098B526DDBAA58BEC3F0C8C9ABD008E8C48020484
8C5A0B0EBF4EE9FFFFB0FCF570C3D3EFC40018FC1590CA0FF99A577F00FEA95E194EB
77B1EEEFF6CAA3B9AA85F9350F6A6B9DE4C97A6FFFEB6F090B98C99B53056347DAC8F
5FA4F2FD32E2D815957B30517990195A81A88B9CABFF5BDC53403B540380043604432730
744888D070900141005308
grestore
1 setgray 1 setlinewidth box stroke
1 xlen div setlinewidth
xlabs
1 ylen div setlinewidth
ylabs
%%Trailer
showpage
restore
save
% Information Page description
/inch { 72 mul } def
initgraphics
erasepage
/Helvetica findfont 12 scalefont setfont
1 inch 10 inch moveto
(Analysis Parameters:) show
1 inch 9 inch moveto
(User:) show
(JimBob)
show
1 inch 8.75 inch moveto
(Input File:) show
(Experiment-1.dat)
show
1 inch 8.5 inch moveto
(Time:) show
(Mon Jan 29 15:06:19 1990)
show
1 inch 8.25 inch moveto
(Seq #:) show
(1)
show

1 inch 7.75 inch moveto
(Hard Copy:-) show
(Print000)

```
show

1 inch 7.25 inch moveto
(Power File:- ) show
(Exp-1.pwr)
show

1 inch 6.75 inch moveto
(Process Parameters:-) show
1 inch 6.25 inch moveto
(File Start: ) show
(11:26:19          )
show
4.5 inch 6.25 inch moveto
(Analysis Start: ) show
(11:26:19          )
show
1 inch 6 inch moveto
(Time per FFT: ) show
(00:00:04.000      )
show
4.5 inch 6 inch moveto
(# FFT's Averaged: ) show
(    1)
show
1 inch 5.75 inch moveto
(Averaging Time: ) show
(00:00:04.000      )
show
4.5 inch 5.75 inch moveto
(Set center sep.: ) show
(00:00:02.000      )
show
1 inch 5.5 inch moveto
(Total Time: ) show
(00:03:20.003)
show
4.5 inch 5.5 inch moveto
(# Sets to do: ) show
(  100)
show
1 inch 5.25 inch moveto
(Data pts/FFT: ) show
(  1024)
show
4.5 inch 5.25 inch moveto
(Zero pad pts: ) show
(    0)
show
1 inch 5. inch moveto
(Window: ) show
(SQUARE  )
show
4.5 inch 5. inch moveto
```

```
(Percent Overlap: ) show
(  50.0000)
show
1 inch 4.75 inch moveto
(Mean Removal: ) show
(NO)
show
4.5 inch 4.75 inch moveto
(Bins Avg'd: ) show
(  1)
show
1 inch 4.5 inch moveto
(Strt/Stop bins: ) show
(  1  512)
show
4.5 inch 4.5 inch moveto
(Min/Max Freqs: ) show
(  0.00  127.75)
show
1 inch 4.25 inch moveto
(Input Chan: ) show
(1-4)
show
4.5 inch 4.25 inch moveto
(Greyscaled Chans: ) show
( 1)
show
1 inch 4. inch moveto
(Input H/P: ) show
(0-1,8-9)
show
4.5 inch 4. inch moveto
(Greyscaled H/P: ) show
( 0)
show
1 inch 3.75 inch moveto
(Normalizer: ) show
(SPLIT_WINDOW)
show
4.5 inch 3.75 inch moveto
(Greyscaling: ) show
(ON)
show
1 inch 3.5 inch moveto
(Grey: ) show
(ON)
show
4.5 inch 3.5 inch moveto
(# Levels: ) show
(  16)
show
4.5 inch 3.25 inch moveto
(Increment: ) show
(  1.0000)
```

```
show
1 inch 3.25 inch moveto
(Black: ) show
(    8.0000)
show
1 inch 3. inch moveto
(Channel Displayed: ) show
(    1)
show
4.5 inch 3. inch moveto
(H/P Displayed: ) show
(    0)
show
1 inch 2.75 inch moveto
(Resolution: ) show
(    0.2500)
show
showpage
%%TRAILER
restore
```

REFERENCES

1. PostScript Language Reference Manual, Adobe Systems Incorporated. Addison-Wesley Publishing Company Ltd., 1985.

2. PostScript Language Tutorial and Cookbook, Adobe Systems Incorporated. Addison-Wesley Publishing Company Ltd., 1985.

3. PostScript Language Program Design, Adobe Systems Incorporated. Addison-Wesley Publishing Company Ltd., 1988.

4. Real World PostScript, Stephen J. Roth. Addison-Wesley Publishing Company Ltd., 1988.

5. Saspex a versatile tool for spectral analysis and display. J.B. Farrell, G.J. Heard, DREA Technical Communication DRAFT.

Figure 1. Grey-scale levels specified by the transarray matrix with the program requested screen frequency and spot function.

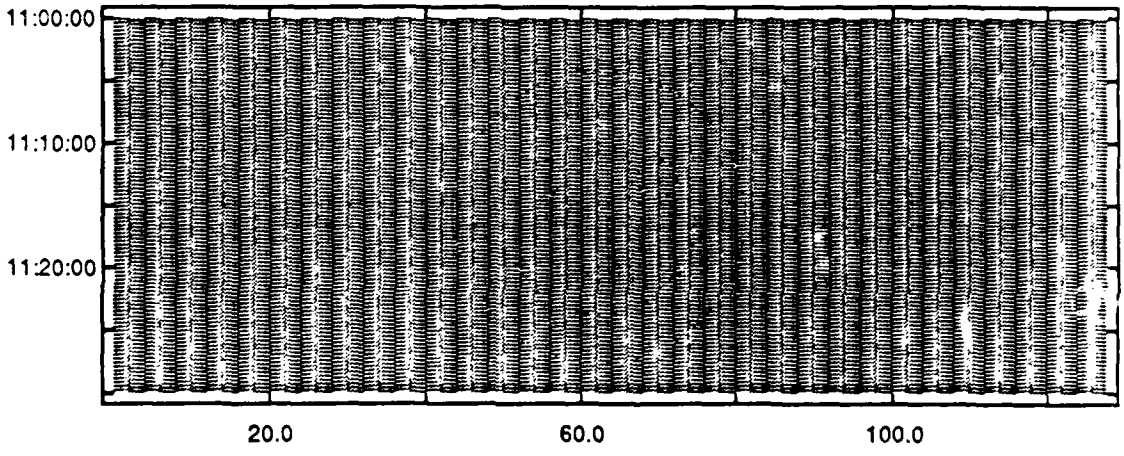

Figure 2. Example of grey-scaled diagram produced with the code described in this report. Arbitrary y and x axis values have been chosen for illustration purposes.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence Research Establishment Atlantic
P.O.Box 1012, Dartmouth, N.S. B2Y 3Z7

2. SECURITY CLASSIFICATION (overall security classification of the document, including special warning terms if applicable)

UNCLASSIFIED

3 TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C,R or U) in parentheses after the title.)

Post Script Code for Grey-Scaled Displays

4. AUTHORS (Last name, first name, middle initial. If military, show rank, e.g. Doe, Maj. John E.)

Heard,G.J.

5. DATE OF PUBLICATION (month and year of publication of document)

April 1990

6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc.) 28

6b. NO. OF REFS (total cited in document) 5

6. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Communication

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.)

DREA

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant)

Project DRDA 06

9b. CONTRACT NO. (if appropnate, the applicable number under which the document was written)

10a. ORIGINATOR 'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)

DREA Technical Communication 90/304

10b. OTHER DOCUMENT NOS. (any other numbers which may be assigned this document either by the originator or by the sponsor)

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(X) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

12.. DOCUMENT ANNOUNCEMENT (any limitations to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where futher distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

No limitation

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

PostScript is a versatile page description language used for the production of complex images combining graphics and text information. This note describes a PostScript program capable of producing a detailed grey-scaled image of information represented by a series of hexadecimal codes. A parameter section is included in the PostScript code that allows the user to modify the appearance of the diagram without the need to re-execute the program that generated the data and/or PostScript file in the first place. This code can be used on many different systems that support a PostScript printing device, but was originally intended to be printed on a 300 dot per inch laser printer. The PostScript code contains procedures to create time strings in hours, minutes, and seconds format from an input value in decimal hours.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is requried. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

PostScript
sonograms
grey-scale
grey-scaled
software
laserprinter
computer
Macintosh
VAX
UNIX